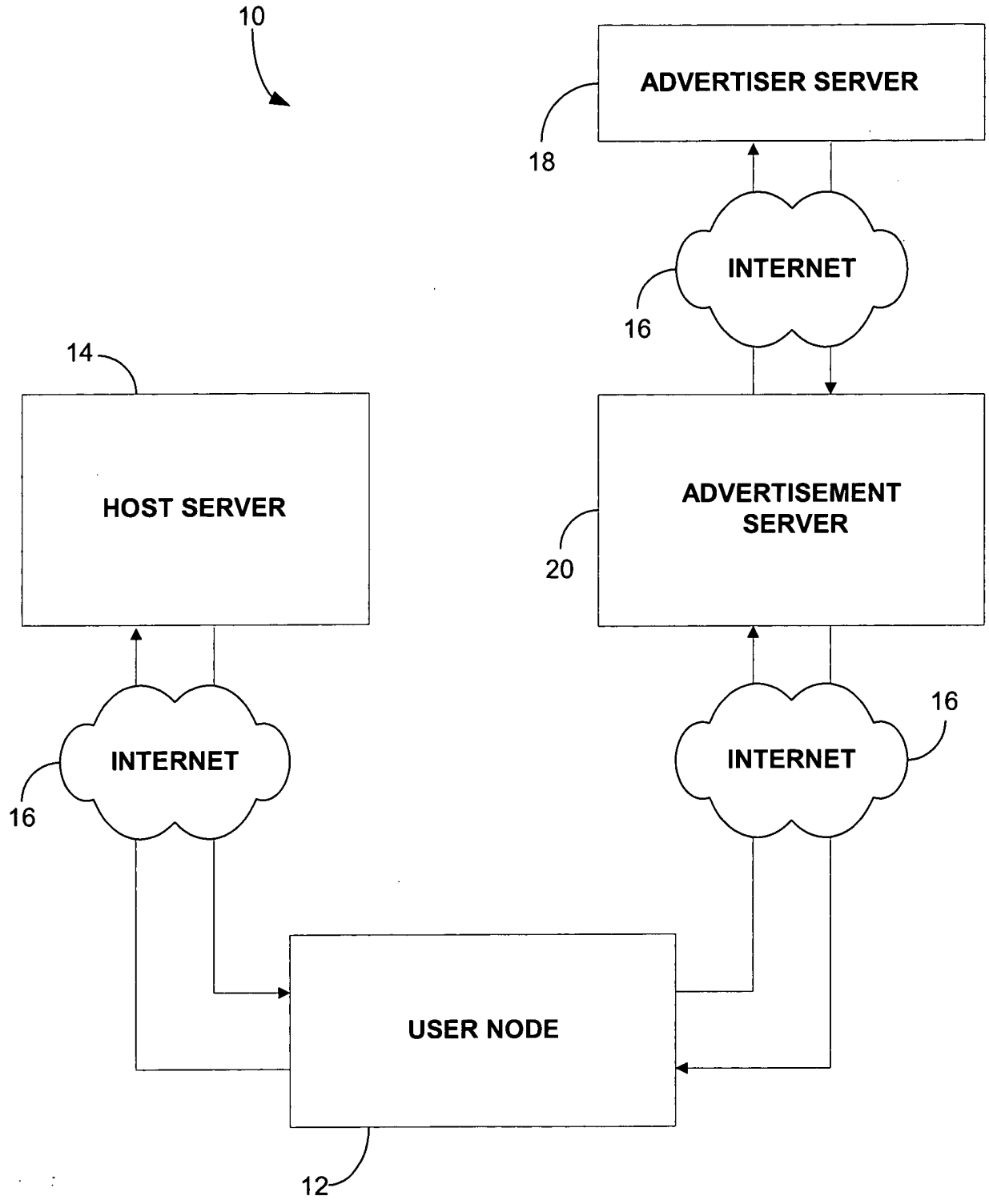
10
ADVERTISER SERVER
18
INTERNET
16
14
HOST SERVER
ADVERTISEMENT SERVER
20
INTERNET
16
INTERNET
16
USER NODE
12

Figure 1

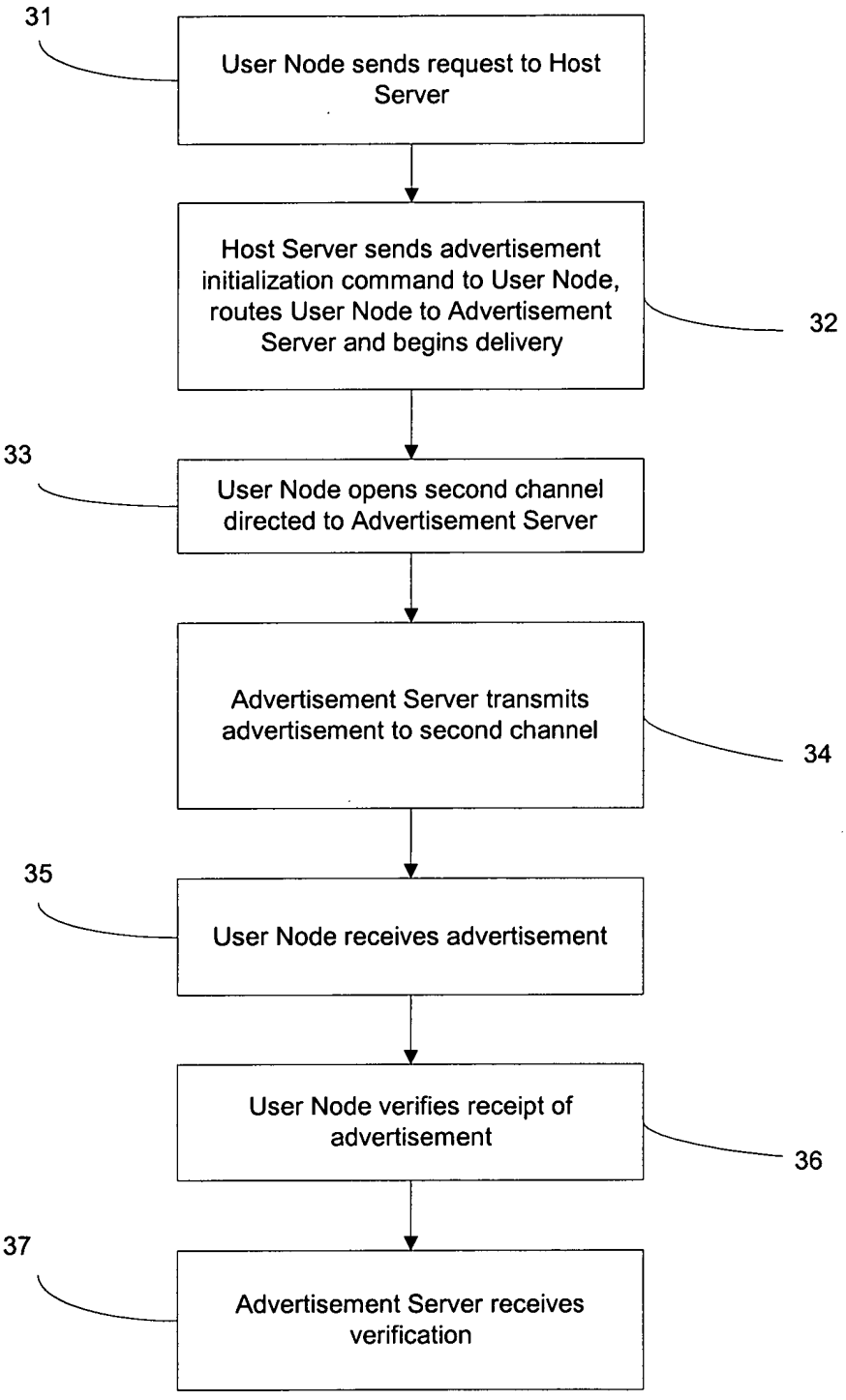
31
User Node sends request to Host Server
30
Host Server sends advertisement initialization command to User Node, routes User Node to Advertisement Server and begins delivery
32
33
User Node opens second channel directed to Advertisement Server
Advertisement Server transmits advertisement to second channel
34
35
User Node receives advertisement
User Node verifies receipt of advertisement
36
37
Advertisement Server receives verification

Figure 2

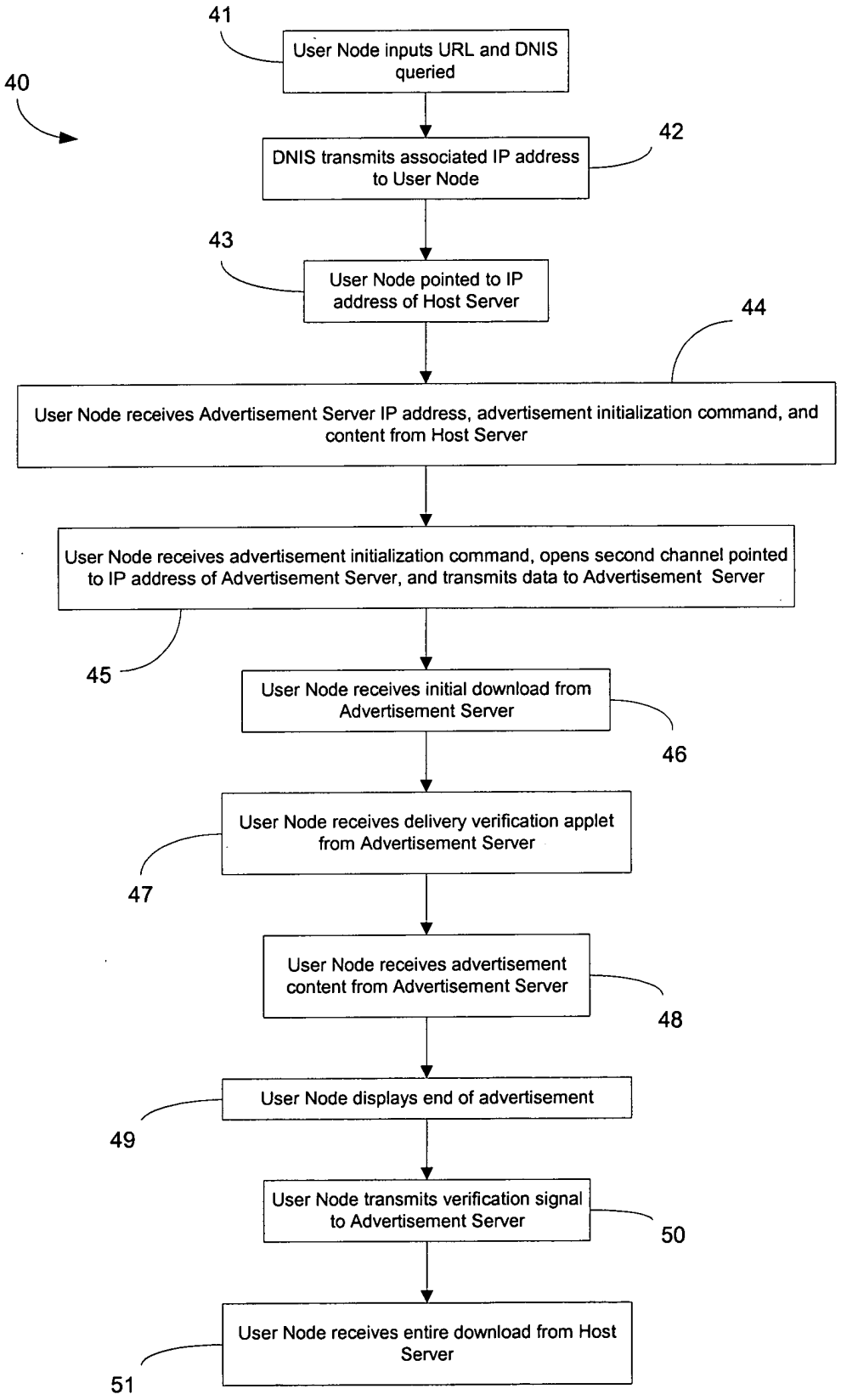
40
41
User Node inputs URL and DNIS queried
42
DNIS transmits associated IP address to User Node
43
User Node pointed to IP address of Host Server
44
User Node receives Advertisement Server IP address, advertisement initialization command, and content from Host Server
User Node receives advertisement initialization command, opens second channel pointed to IP address of Advertisement Server, and transmits data to Advertisement Server
45
User Node receives initial download from Advertisement Server
46
User Node receives delivery verification applet from Advertisement Server
47
User Node receives advertisement content from Advertisement Server
48
User Node displays end of advertisement
49
User Node transmits verification signal to Advertisement Server
50
User Node receives entire download from Host Server
51

Figure 3

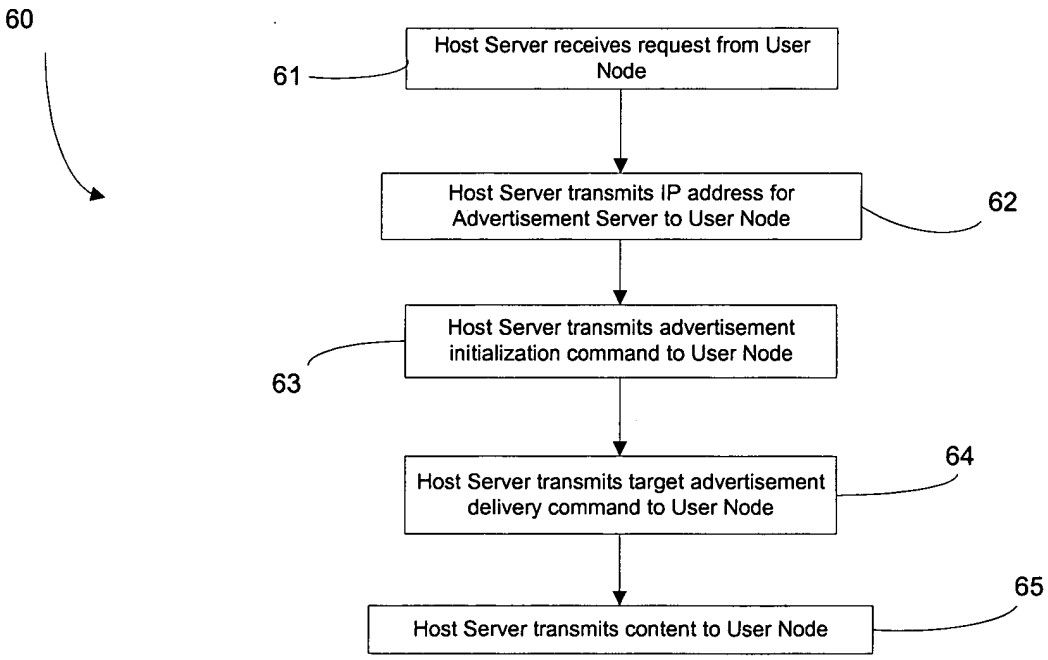
60
61
Host Server receives request from User Node
Host Server transmits IP address for Advertisement Server to User Node
62
Host Server transmits advertisement initialization command to User Node
63
Host Server transmits target advertisement delivery command to User Node
64
Host Server transmits content to User Node
65

Figure 4

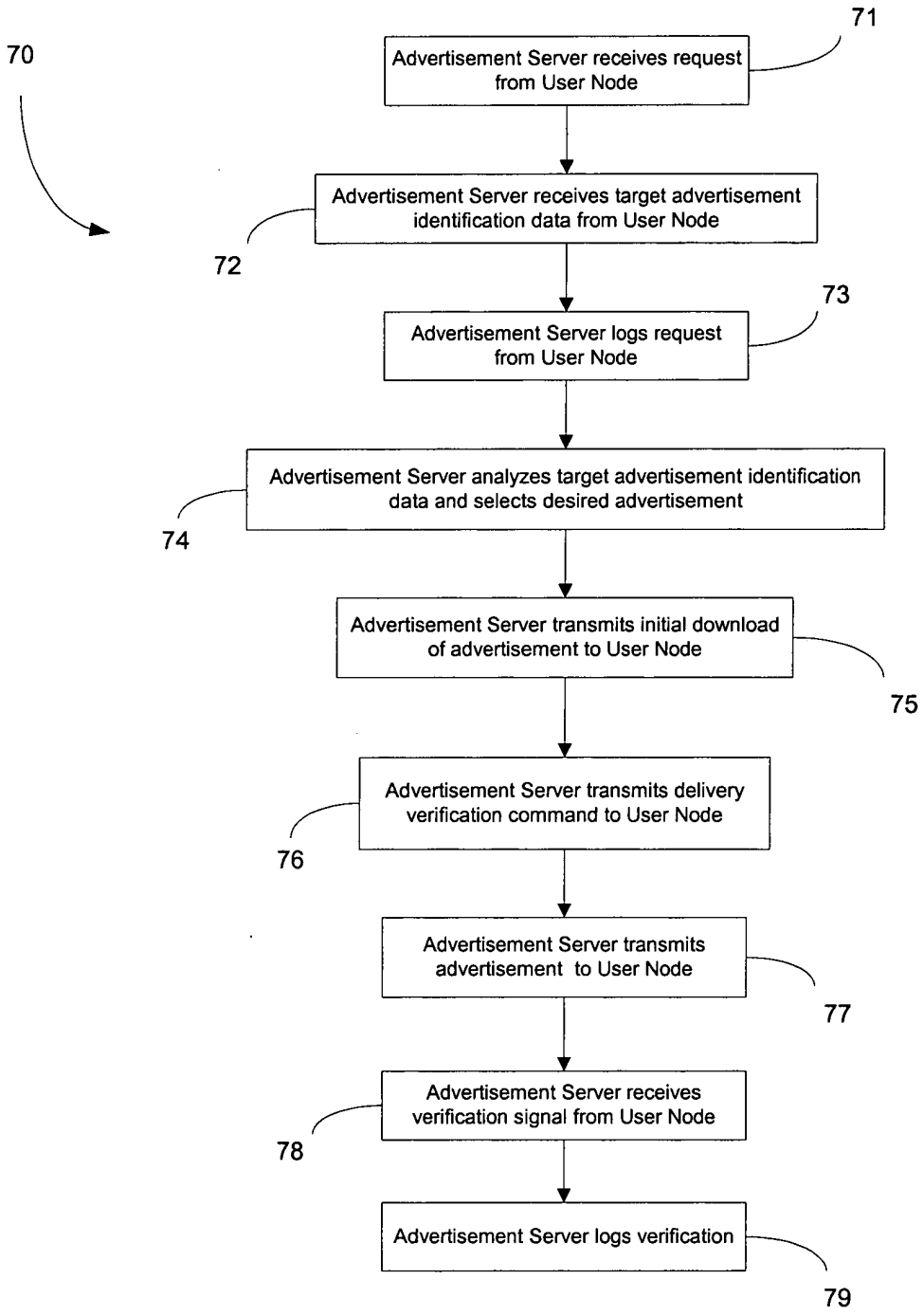
70
71
Advertisement Server receives request from User Node
Advertisement Server receives target advertisement identification data from User Node
72
73
Advertisement Server logs request from User Node
Advertisement Server analyzes target advertisement identification data and selects desired advertisement
74
Advertisement Server transmits initial download of advertisement to User Node
75
Advertisement Server transmits delivery verification command to User Node
76
Advertisement Server transmits advertisement to User Node
77
Advertisement Server receives verification signal from User Node
78
Advertisement Server logs verification
79

Figure 5

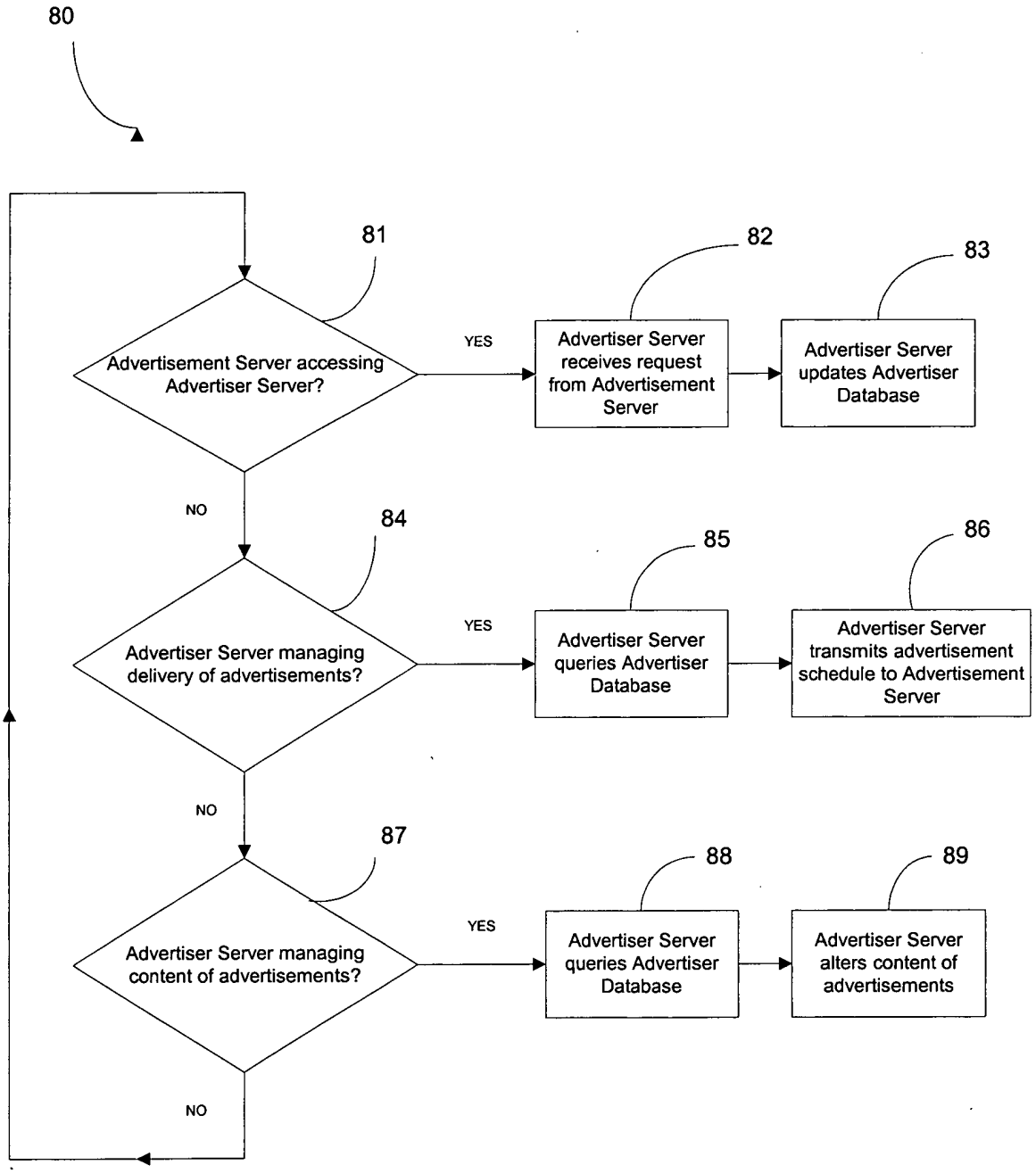
80
81
Advertisement Server accessing Advertiser Server?
YES
82
Advertiser Server receives request from Advertisement Server
83
Advertiser Server updates Advertiser Database
NO
84
Advertiser Server managing delivery of advertisements?
YES
85
Advertiser Server queries Advertiser Database
86
Advertiser Server transmits advertisement schedule to Advertisement Server
NO
87
Advertiser Server managing content of advertisements?
YES
88
Advertiser Server queries Advertiser Database
89
Advertiser Server alters content of advertisements
NO

Figure 6

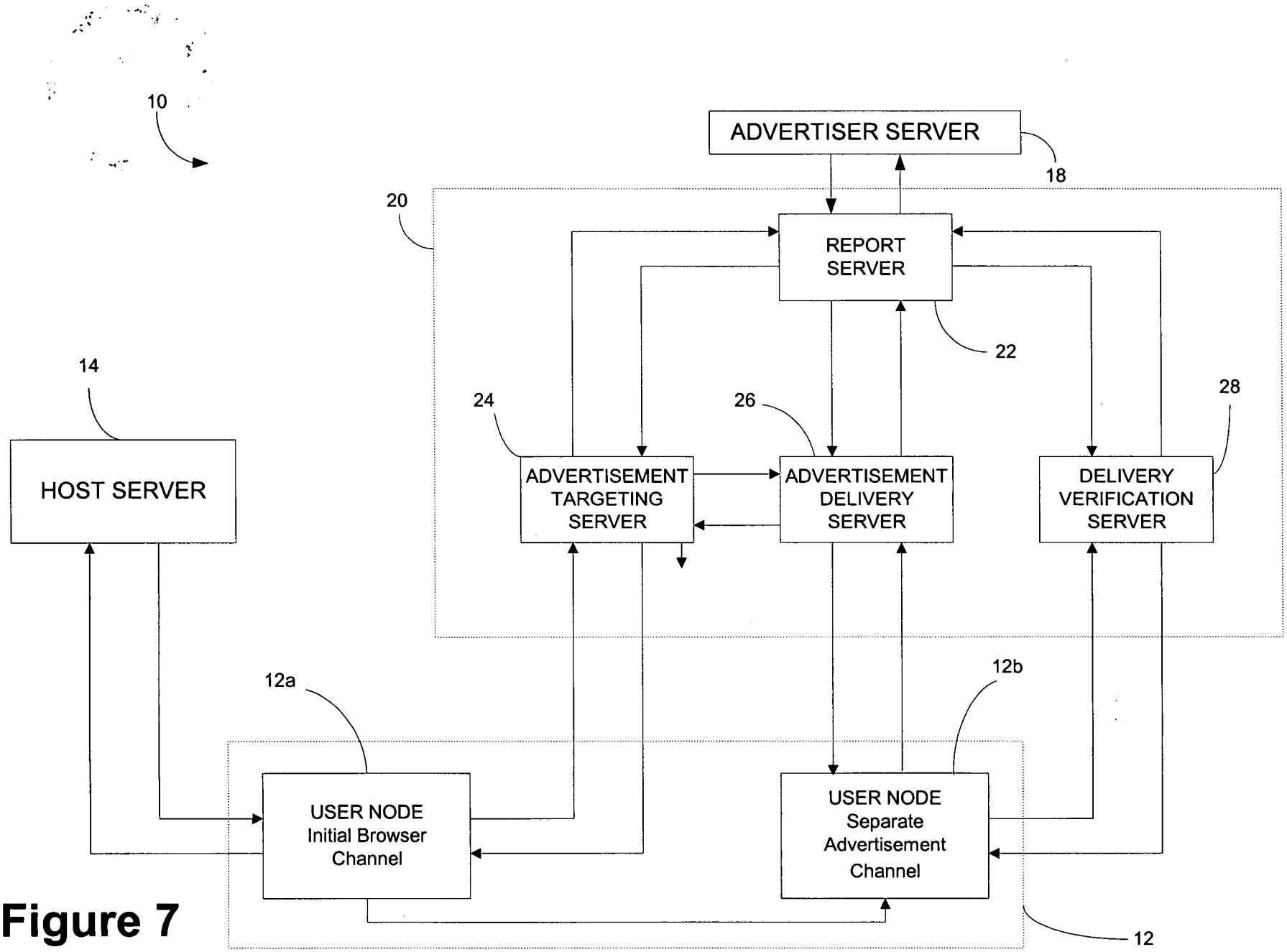

Figure 7

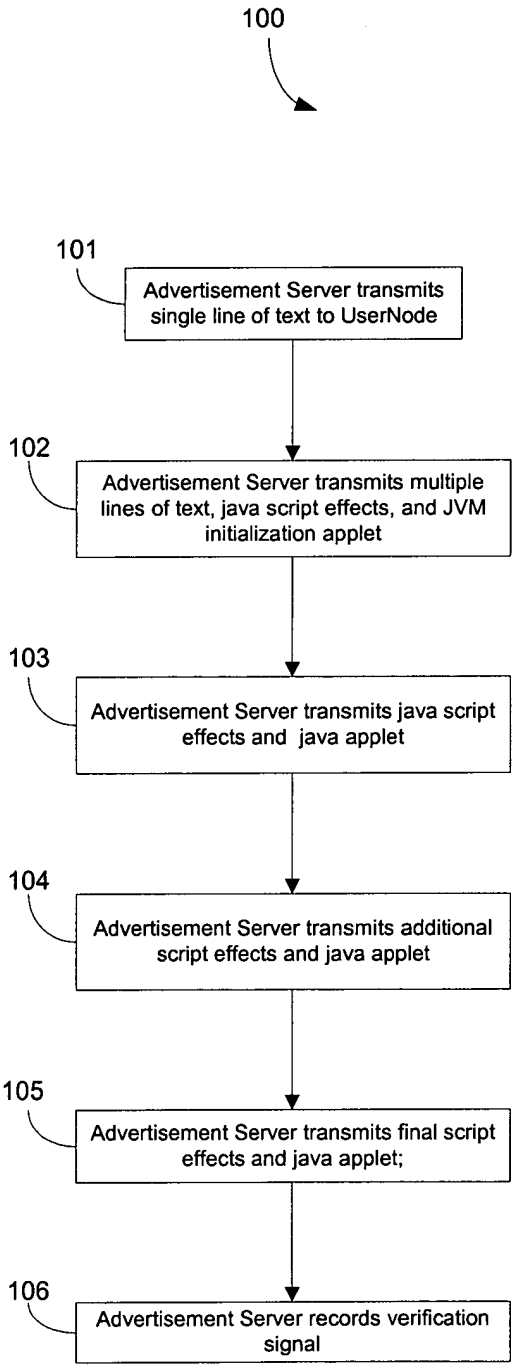

Figure 8a

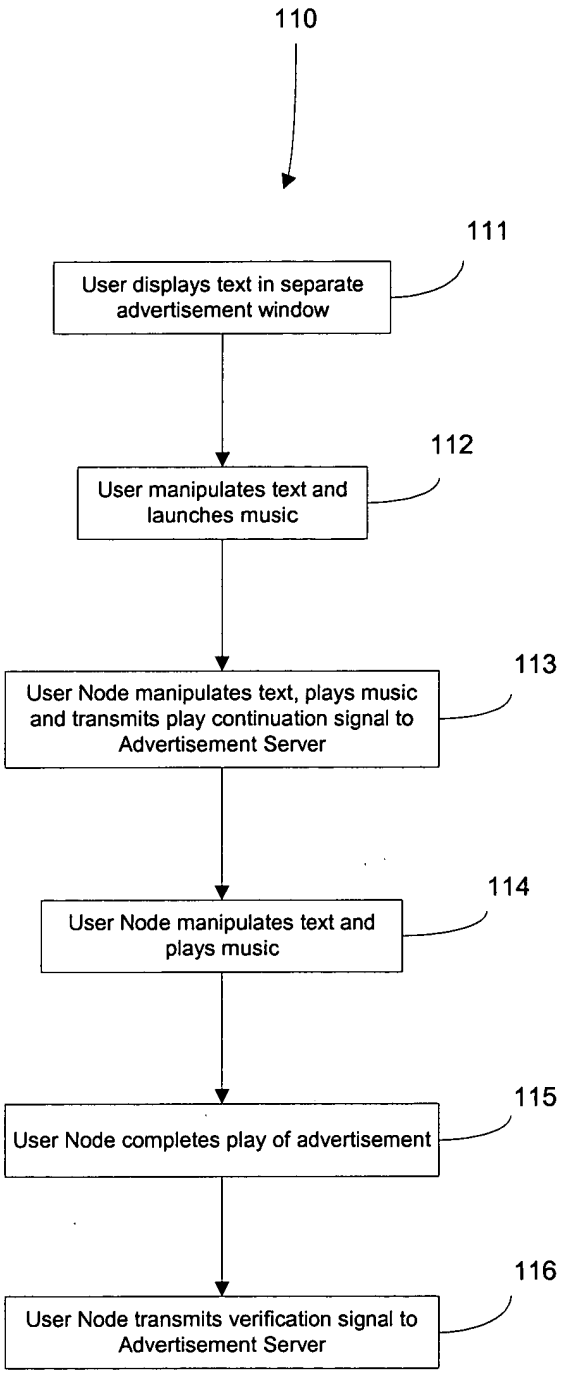

Figure 8b

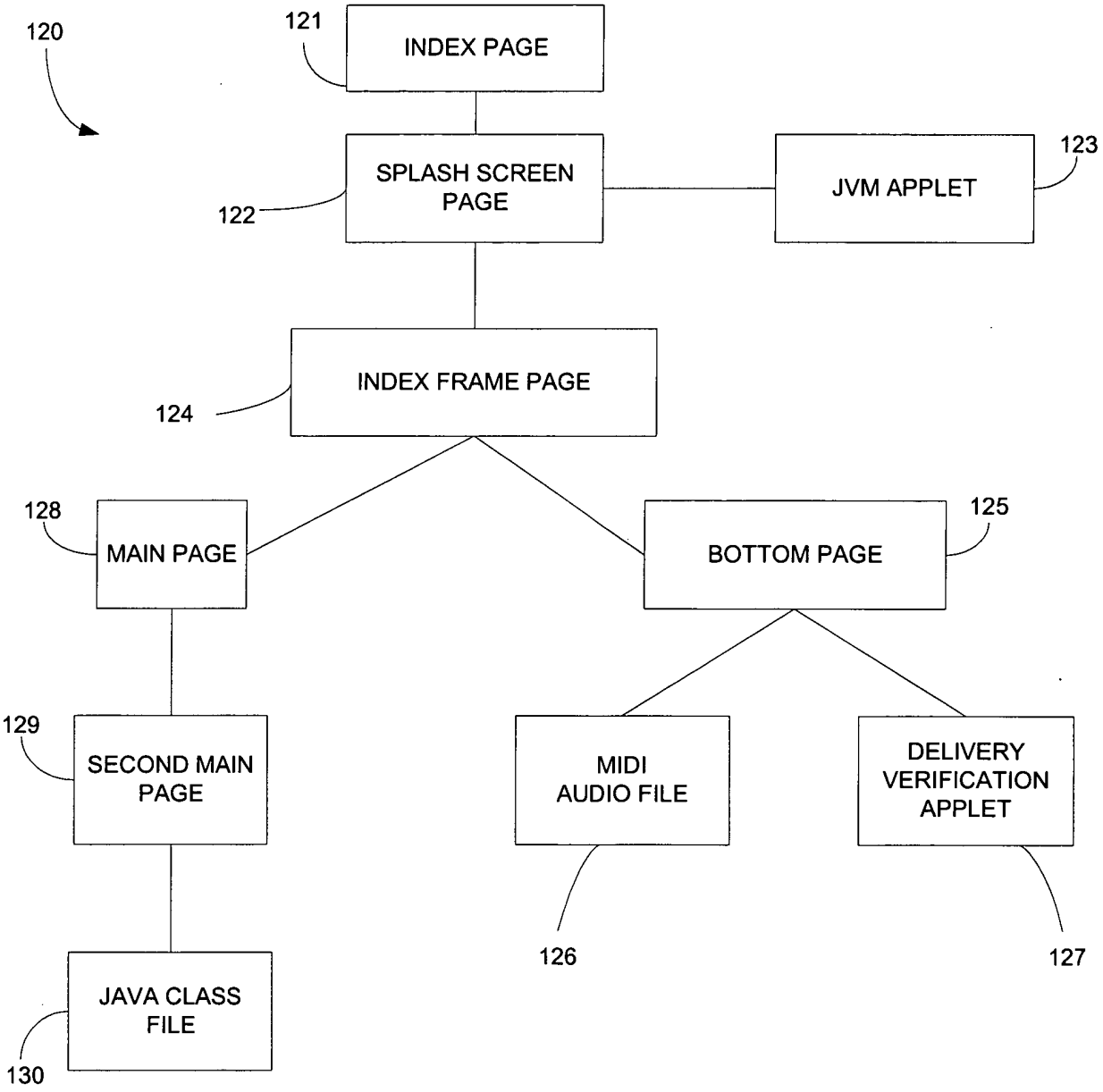
120
121
INDEX PAGE
SPLASH SCREEN PAGE
122
JVM APPLET
123
INDEX FRAME PAGE
124
128
MAIN PAGE
BOTTOM PAGE
125
129
SECOND MAIN PAGE
MIDI AUDIO FILE
DELIVERY VERIFICATION APPLET
126
127
JAVA CLASS FILE
130

Figure 9

```
<html>
<head>
<title>Chevy</title>
<meta http-equiv="REFRESH" content="20;URL=http://www.chevy.com">
<script language="JavaScript">
<!--
function nrwmwbndw(theURL,winName,features) { //v2.0
window.open(theURL,winName,features);
}
//-->
</script>
</head>
<body bgcolor="#ffffff"
onLoad="nrwmwbndw('indexa.html','chevy','fullscreen,scrolling=no,border=0')">
<div align="center">
  <p> </p>
  <p> </p>
  <p>The Chevrolet home page will load momentarily</p>
  <p> </p>
</div>
</body>
</html>
```

Figure 10

```
<head>
<meta http-equiv="REFRESH" content="0;URL=indexframe.html">
<title>Chevrolet Silverado</title>
</head>
<body bgcolor=#000000>

<p> </p>
<p> </p>
<p> </p>

<center>
<font size="7" color="#FF6600" face="Times New Roman, Times, serif">
LIKE A ROCK
</font>

<applet code=jvm.class width=0 height=0></applet>

</center>
```

Figure 11

```
import java.applet.*;
public class jvm extends Applet {
public void init() { }
}
```

Figure 12

```
<title>Chevrolet Silverado</title>

<frameset rows="*,1" border=0>
<frame name=main src=main.html noresize>
<frame name=bottom src=bottom.html noresize>
</frameset>
```

Figure 13

```
<body bgcolor=#000000 bgsound=LIKEROK2.mid>
<embed src=likerok2.mid autostart=true width=0 height=0>
```

Figure 14


```
<meta http-equiv="REFRESH" content="2;URL=main2.html">

<body bgcolor=#000000>

<p> </p>
<p> </p>
<p> </p>

<center>

<b>
<font size="7" color="#FFFFFF">
CHEVY <i>TRUCKS</i></font>
</b>
</font>

</center>
```

Figure 15

```
<head>
<script>
function closeSelf() {
 parent.close();
}

function closer() {
        setTimeout('closeSelf()',35000);
}
</script>

</head>
<body bgcolor=#000000 onload="closer();">

<p> </p>
<br>
<center>
<applet code=template.class width=640 height=480>
</applet>
</center>
```


Figure 16

```
import java.awt.*;
import java.applet.*;


public class tlayer extends Canvas {

String str;
path x,y;
path r,g,b;
path s;
path z;
path spacing;
boolean visible,visibleAtEnd;
int steps;
int ttype;

FontMetrics fm;



public tlayer(String s) {

        str = s;
        visible = true;
        visibleAtEnd = true;

        ttype=Font.PLAIN;

} // tlayer
```


Figure 17a

```
public void steps(int steps) {
        this.steps = steps;
}

public void position(int x1,int y1,int x2,int y2,int steps) {

        x=new path(x1,x2,steps,0);
        y=new path(y1,y2,steps,0);


} // position

public void font(int ttype) {
        this.ttype=ttype;
}

public void spacing(int x1,int x2,int steps) {

        spacing=new path(x1,x2,steps,0);

} // spacing

public void color(int r1,int g1,int b1,int r2,int g2,int b2,int steps) {
        r=new path(r1,r2,steps,0);
        g=new path(g1,g2,steps,0);
        b=new path(b1,b2,steps,0);
} // color

public void size(int s1,int s2,int steps) {
        s=new path(s1,s2,steps,0);
}

public void zindex(int z1,int z2,int steps) {
        z=new path(z1,z2,steps,0);
}
```


Figure 17b

```
public void setVisible(boolean v,boolean vAtEnd) {
        visible = v;
        visibleAtEnd = vAtEnd;
}

public void paintOffscreen(Graphics graphics) {

        // different postions,color,sizes,and whether visible

        if(visible) {

                Color c=new Color(r.getX(),g.getX(),b.getX());
                graphics.setColor(c);

                Font f=new Font("TimesRoman",ttype,s.getX());
                graphics.setFont(f);

                FontMetrics fm=getFontMetrics(f);

                int cx=x.getX();
                int cy=y.getX();
                for(int i=0;i<str.length();i++) {
                        graphics.drawString(""+str.charAt(i),cx,cy);
                        cx+=fm.charWidth(str.charAt(i))+spacing.getX();
                }
//              graphics.drawString(str,x.getX(),y.getX());

        //      System.out.println("x1 = " + x.getX() + " y1 = " + y.getX() + " for " +
c);

        } // visible


} // paintOffscreen
```


Figure 17c

```
public void incTimer() {

        if(steps>0) {

                x.incStep();
                y.incStep();
                r.incStep();
                g.incStep();
                b.incStep();
                s.incStep();
                z.incStep();
                spacing.incStep();

                steps--;

                if(steps==0 && !visibleAtEnd)
                        visible=false;


        }

} // incTimer


} // tlayer
```


Figure 17d

```
import java.awt.*;
import java.applet.*;


public class ilayer extends Canvas {

Image im;
path x,y;
path sx,sy;
path z;
boolean visible,visibleAtEnd;
int steps;



public ilayer(Image im) {

        this.im=im;

        visible = true;
        visibleAtEnd = true;

} // tlayer

public int getX() {
        return x.getX();
}
public int getY() {
        return y.getX();
}
public int getSX() {
        return sx.getX();
}
public int getSY() {
        return sy.getX();
}
```


Figure 18a

```
public boolean getVisible() {
        return visible;
}


public void steps(int steps) {
        this.steps = steps;
}

public void position(int x1,int y1,int x2,int y2,int steps) {

        x=new path(x1,x2,steps,0);
        y=new path(y1,y2,steps,0);


} // position

public void size(int x1,int y1,int x2,int y2,int steps) {

        sx=new path(x1,x2,steps,0);
        sy=new path(y1,y2,steps,0);

} // position



public void zindex(int z1,int z2,int steps) {
        z=new path(z1,z2,steps,0);
}

public void setVisible(boolean v,boolean vAtEnd) {
        visible = v;
        visibleAtEnd = vAtEnd;
}
```


Figure 18b

```
public void paintOffscreen(Graphics graphics) {

        // different postions,color,sizes,and whether visible

        if(visible) {

                graphics.drawImage(im,x.getX(),y.getX(),sx.getX(),sy.getX(),this);

//              System.out.println("x1 = " + x.getX() + " y1 = " + y.getX());

        } // visible


} // paintOffscreen


public void incTimer() {

        if(steps>0) {

                x.incStep();
                y.incStep();
                sx.incStep();
                sy.incStep();

                z.incStep();

                steps--;

                if(steps==0 && !visibleAtEnd)
                        visible=false;


        }

} // incTimer

} // ilayer
```

Figure 18c

```
public class path {

 double x1,x2;
 double xstep;
 int eq,steps;


public path(double x1,double x2,int steps,int eq) {

        this.x1=x1;
        this.x2=x2;

        this.steps=steps;
        this.eq=eq;

        xstep=(x2-x1)/(double) steps;
//      System.out.println("**** init x1 = " + x1 + "  x2 = " + x2 + " step = " + xstep);


} // path constructor


public void incStep() {

        x1=x1+xstep;

} // incStep

public int getX() {
        return (int) x1;
}


} // path class
```

Figure 19

```
/*
   Java DVT Client --> connecting to --> Multi-threaded DVT C Server

*/

import java.awt.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.applet.*;


public class dvtclient extends Applet implements Runnable {

 Socket clientSocket=null;
 DataInputStream dis=null;
 OutputStream os=null;
 String host;
 int port;
 int delay;
 boolean isRunning=true;
 int messages=0;

 Thread threadRef=null;


public void init() {

 System.out.println("starting constructor");
 host="38.202.155.30";
 port=2048;
 delay=1000;

 System.out.println("Done with constructor");

 // create socket communications
 System.out.println("Attemping to connect to port "+host+":"+port+"\n");
```

Figure 20a

```
try {
        clientSocket=new Socket(host,port);
}catch(Exception makingsocket) {
        System.out.println("Error connecting to " + host + " at port " + port);
        return;
        }

System.out.println("Made Connection...");


try {
  dis=new DataInputStream(clientSocket.getInputStream());
  os=clientSocket.getOutputStream();

  }catch(UnknownHostException e) {
    System.out.println("Unknown Host exeception getting socket streams!!!");
   }catch(IOException e) {
     System.out.println("IO exeception getting socket streams!!!");
   }

System.out.println("Made input/output connections");



threadRef = new Thread(this);
threadRef.start();


} // constructor
```


Figure 20b

```
public synchronized void start() {

 if(threadRef==null) {
        System.out.println("Null threadRef in start()");
        threadRef = new Thread(this);
        threadRef.start();
 }

} // start


public void stop() {

  System.out.println("Stopping...");
  if(threadRef!=null) {

    threadRef.stop();

    threadRef=null;
    }
} // stop

public void destroy() {

  System.out.println("Destroying...");
  if(threadRef!=null) {

    threadRef.stop();

    threadRef=null;
    }
} // destroy
```

Figure 20c

```
public void run() {


 System.out.println("Starting run..");
 while(isRunning && messages<3) {

        SendAndReceive();
        messages++;
        System.out.println("Going to sleep for 1 second");
        try {
                threadRef.sleep(delay);
        }catch(Exception e) {
                System.out.println("Error in sleep");
        }

 } // while isrunning

 try {
         System.out.println("Closing socket...");
         clientSocket.close();
         stop();
         destroy();
 }catch(Exception e) {
        System.out.println("Error Closing socket");
 }



} // run



void SendAndReceive() {

 byte bbuf[]=new byte[256];
 String str;

 try {
```


Figure 20d

```
System.out.println("command being sent.");
str="message "+messages;
for(int i=0;i<str.length();i++)
        bbuf[i]=(byte) str.charAt(i);
bbuf[str.length()]='\0';
os.write(bbuf,0,str.length());
os.flush();
System.out.println("Command written to server");



// read string back
str= dis.readLine();


System.out.println("Got: " + str);

if(str.length()==0) {
  System.out.println("ERROR receiving from " + host + ":" + port);
  clientSocket.close();
  return;
  }


}catch(Exception e) {
   System.out.println("Exception during send/receive");
}


} // SendAndReceive

} // dvtclient
```

Figure 20e

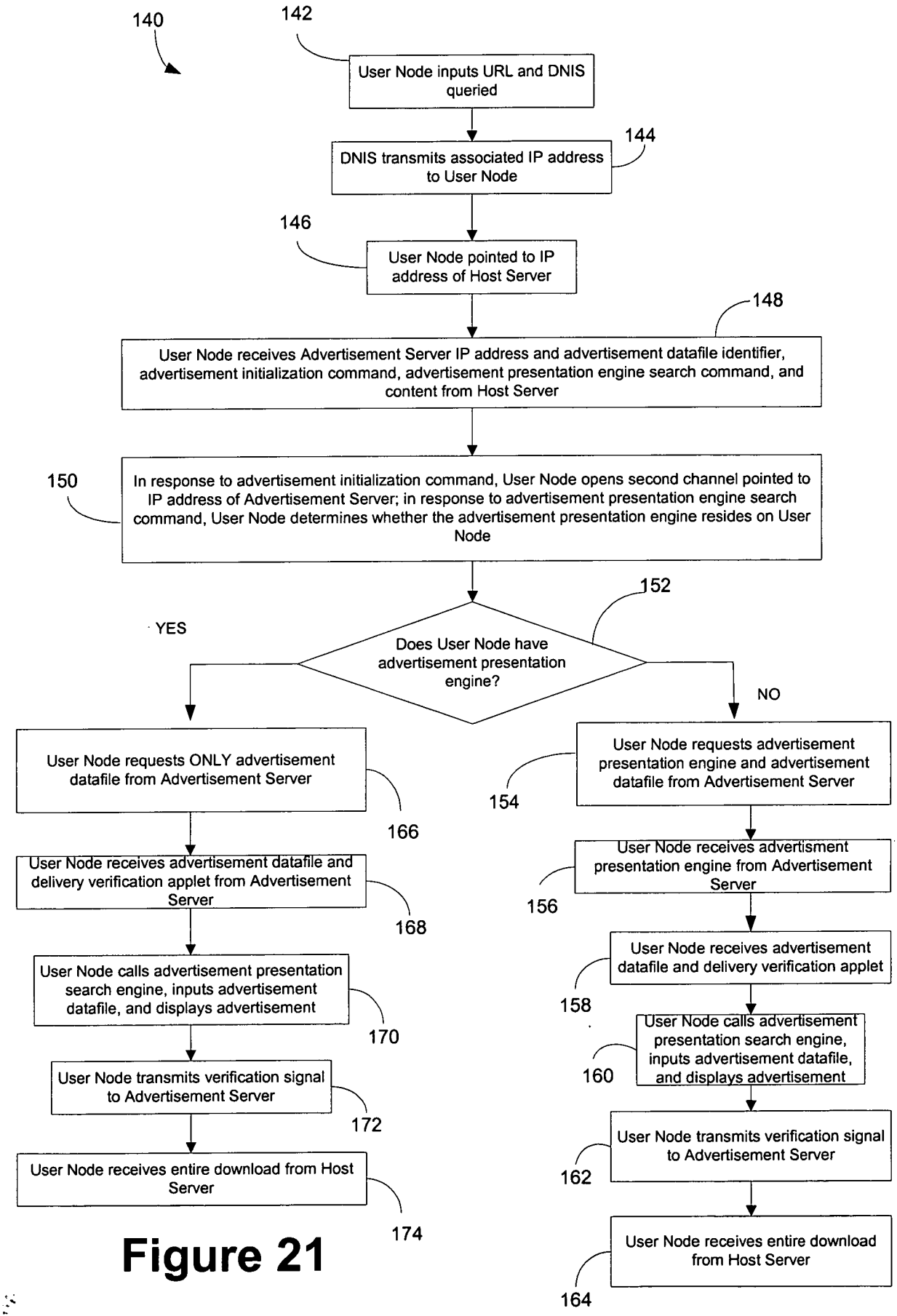
140
142
User Node inputs URL and DNIS queried
144
DNIS transmits associated IP address to User Node
146
User Node pointed to IP address of Host Server
148
User Node receives Advertisement Server IP address and advertisement datafile identifier, advertisement initialization command, advertisement presentation engine search command, and content from Host Server
150
In response to advertisement initialization command, User Node opens second channel pointed to IP address of Advertisement Server; in response to advertisement presentation engine search command, User Node determines whether the advertisement presentation engine resides on User Node
152
Does User Node have advertisement presentation engine?
YES
NO
User Node requests ONLY advertisement datafile from Advertisement Server
166
User Node receives advertisement datafile and delivery verification applet from Advertisement Server
168
User Node calls advertisement presentation search engine, inputs advertisement datafile, and displays advertisement
170
User Node transmits verification signal to Advertisement Server
172
User Node receives entire download from Host Server
174
User Node requests advertisement presentation engine and advertisement datafile from Advertisement Server
154
User Node receives advertisment presentation engine from Advertisement Server
156
User Node receives advertisement datafile and delivery verification applet
158
User Node calls advertisement presentation search engine, inputs advertisement datafile, and displays advertisement
160
User Node transmits verification signal to Advertisement Server
162
User Node receives entire download from Host Server
164

Figure 21

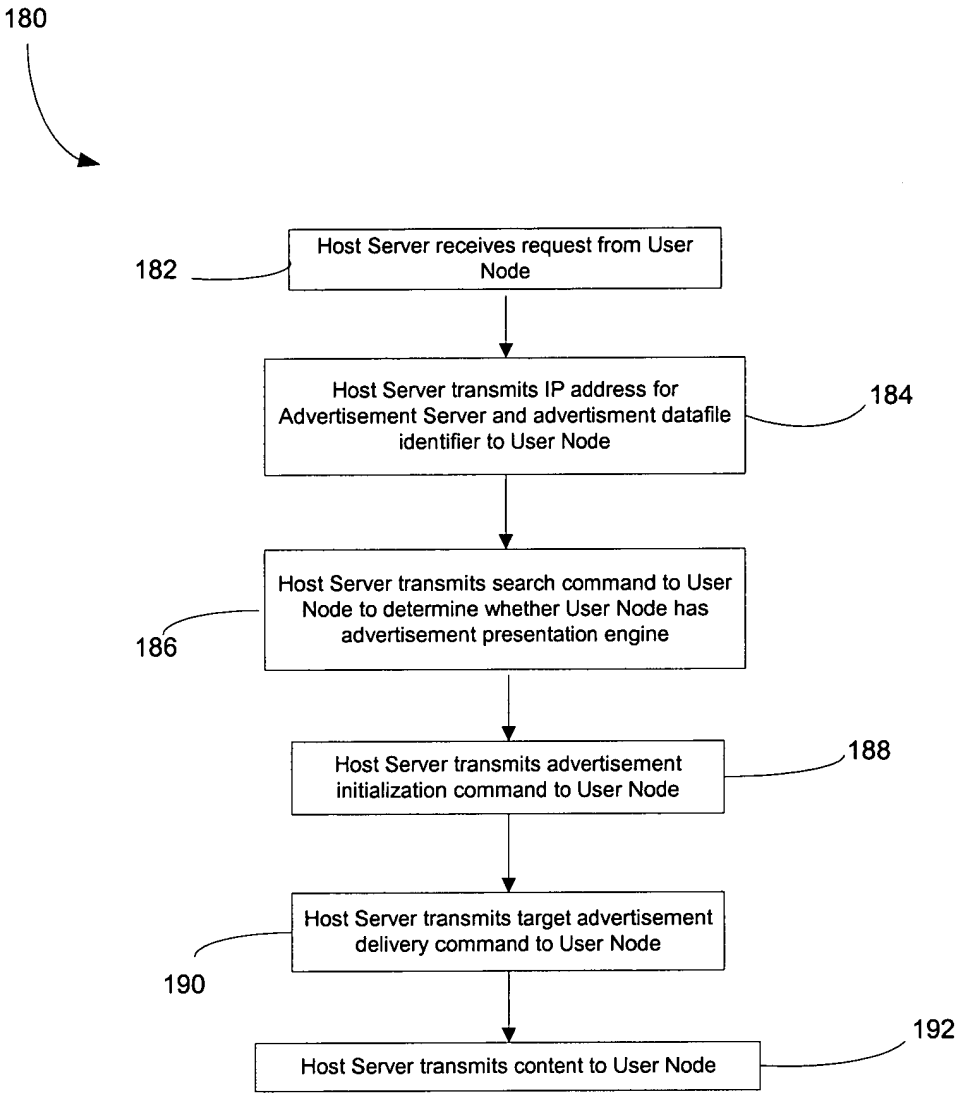
180
182
Host Server receives request from User Node
Host Server transmits IP address for Advertisement Server and advertisment datafile identifier to User Node
184
Host Server transmits search command to User Node to determine whether User Node has advertisement presentation engine
186
Host Server transmits advertisement initialization command to User Node
188
Host Server transmits target advertisement delivery command to User Node
190
Host Server transmits content to User Node
192

Figure 22

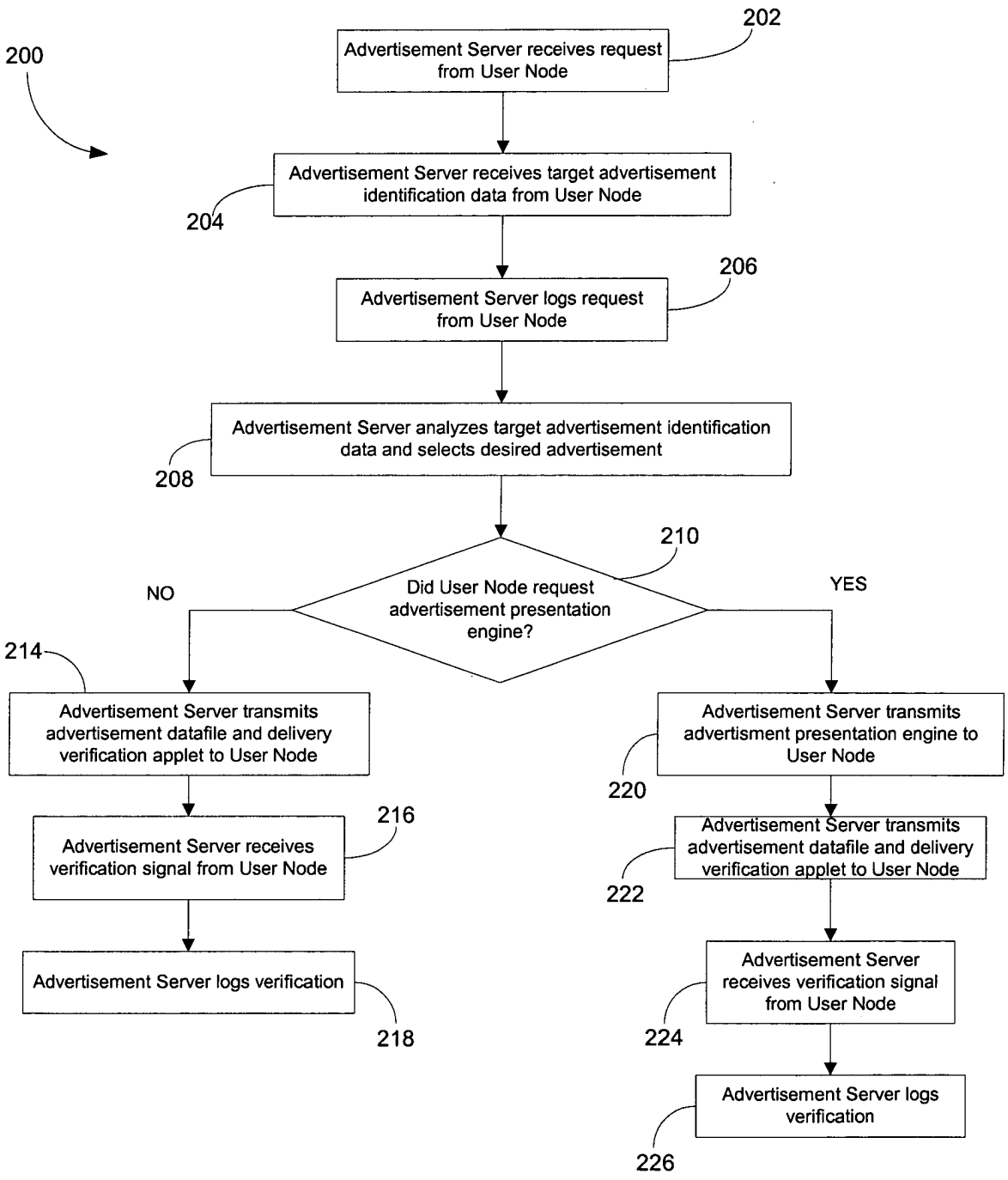
200
Advertisement Server receives request from User Node
202
Advertisement Server receives target advertisement identification data from User Node
204
Advertisement Server logs request from User Node
206
Advertisement Server analyzes target advertisement identification data and selects desired advertisement
208
Did User Node request advertisement presentation engine?
210
NO
YES
Advertisement Server transmits advertisement datafile and delivery verification applet to User Node
214
Advertisement Server receives verification signal from User Node
216
Advertisement Server logs verification
218
Advertisement Server transmits advertisment presentation engine to User Node
220
Advertisement Server transmits advertisement datafile and delivery verification applet to User Node
222
Advertisement Server receives verification signal from User Node
224
Advertisement Server logs verification
226

Figure 23

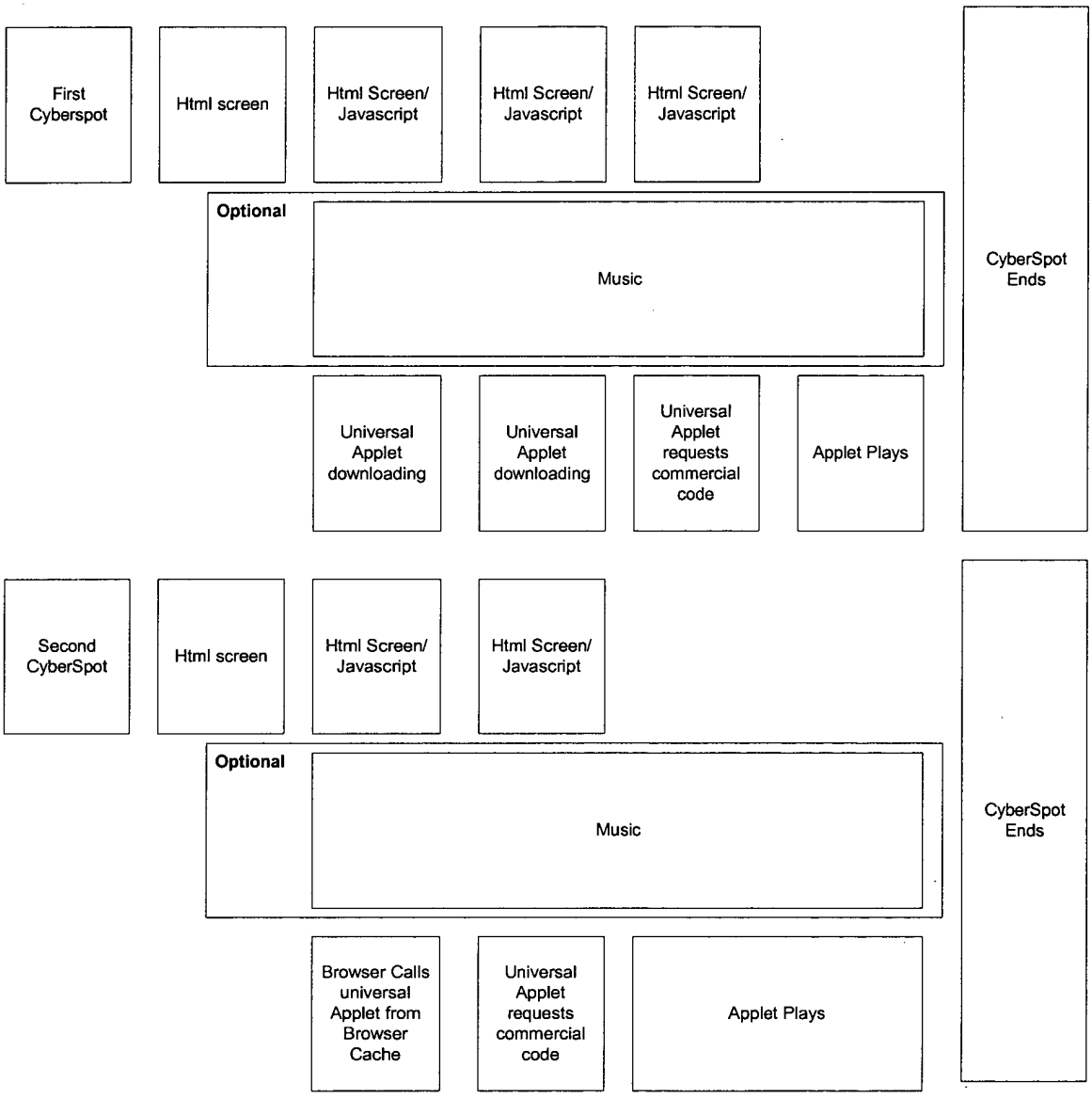
First Cyberspot
Html screen
Html Screen/ Javascript
Html Screen/ Javascript
Html Screen/ Javascript
CyberSpot Ends
Optional
Music
Universal Applet downloading
Universal Applet downloading
Universal Applet requests commercial code
Applet Plays
Second CyberSpot
Html screen
Html Screen/ Javascript
Html Screen/ Javascript
CyberSpot Ends
Optional
Music
Browser Calls universal Applet from Browser Cache
Universal Applet requests commercial code
Applet Plays

Figure 24(a)

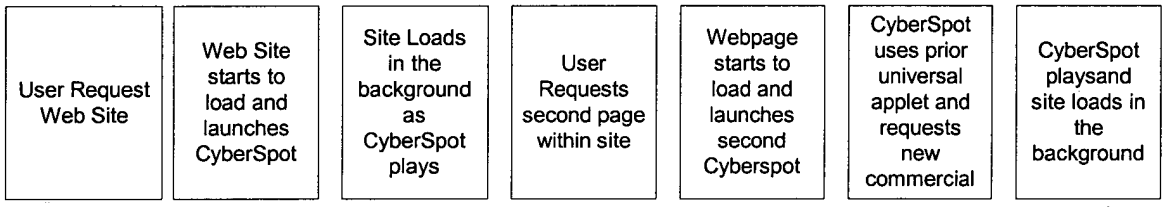

Figure 24(b)

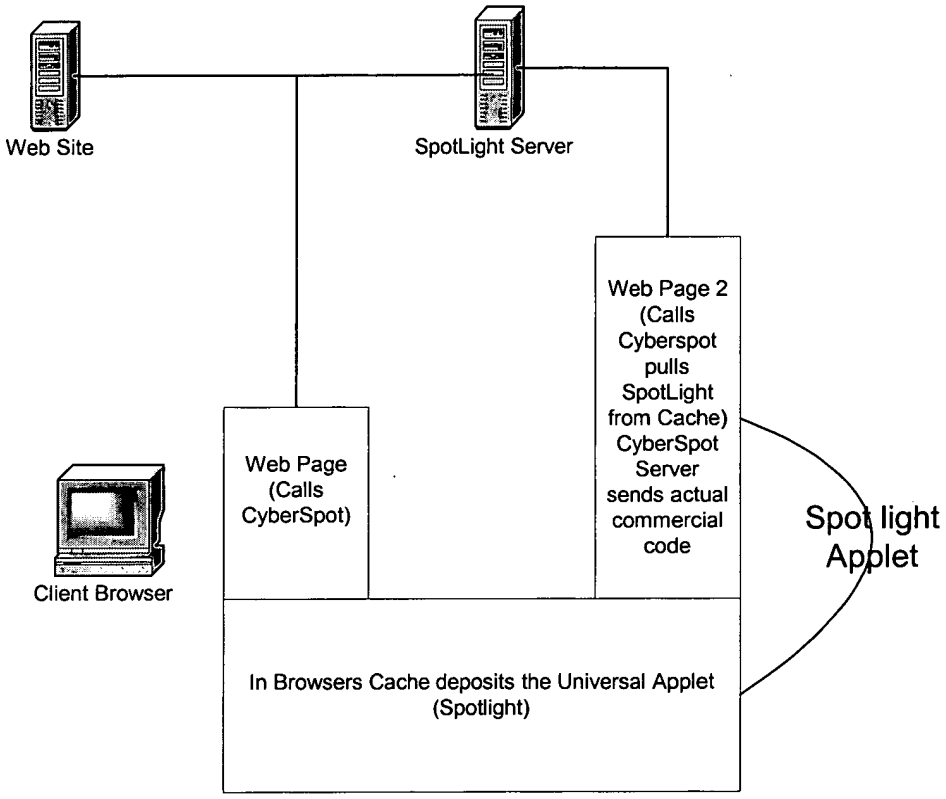
Web Site
SpotLight Server
Web Page 2 (Calls Cyberspot pulls SpotLight from Cache) CyberSpot Server sends actual commercial code
Web Page (Calls CyberSpot)
Client Browser
Spot light Applet
In Browsers Cache deposits the Universal Applet (Spotlight)

Figure 25